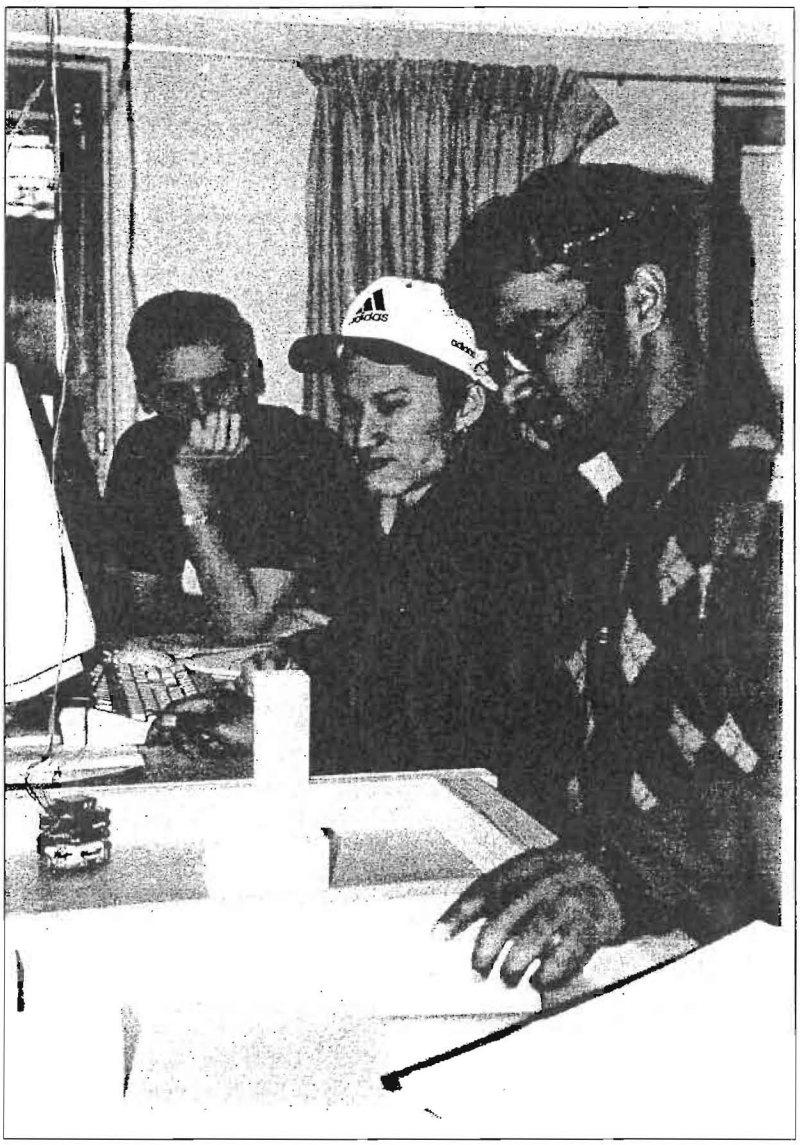
AI Robot Project Report

DAT125, AI for Robots, HT-00



Project 4
group C1: C2

Bo Do
Jeton Grajqevci
Robert Hedlerfog

Contents

1. Introductory.....3
1.1 Method.....3

2. Project description.....3
3. Continuous working process of the robot project.....4
3.1 Planning phase.....5
3.1 Introduction phase.....5
3.2 Implementation phase.....6

4. Problems, solutions and constraints.....10

5. Results and the final status of the robot.....11
6. Final Analysis.....12
7. Future work on the robot.....12
8. Opinions.....13
9 References.....13

Appendix 14

1 Introductory

This is a report of our robot project work in course DAT125, *AI for Robots, HT-00.*
Our group C1, consisting of Bo Do, Jeton Grajqevci and Robert Hedlerfog, chose project no 4 with the following definition:
Hide and seek - looking in a maze for a robot holding a light source on it

We decided to extend the project if time would allow us. The extension was to challenge another group and implement the ability to track another robot and then move home as quickly as possible.

1.1 Method

We decided to use the JAVA language since we all had some knowledge and interest in Java programming. The communication protocol written by Robert Pallbo in Java was another argument for us to complete the project with JAVA. We decided to use the reactive robot paradigm. This decision was made when we realised that the sensors were useful and efficient enough to use this strategy. We also had in mind from the lectures that this way may be easier then an integrated programming paradigm. Parallel testing and programming of the robot was our main strategy to complete our task.

2 Project description

There were three levels of tasks that were included in this project. The base goal was the task all groups were required to implement. The main goal was the task definition of the project our group chose. The additional task was the extra work we wanted to do with our robot in case we had enough time.

Base goal: (also called the introductory phase)

"The introductory phase consists of forcing the robot to explore the environment and create its map. You are free to choose the type of the map you wish your robot to use. What is important is that your robot may handle simple labyrinths set up by an external observer. You have to explain here the assumptions you make and make sure that you understand when they fail."

"The next step is to let the robot locate itself using the map and to find the home position defined by a landmark (possibly a bar-code attached to the place)."

Main goal :

To implement Hide and seek behaviour where the robot is supposed to look in a maze for a robot holding a light source on it.

Additional goal:

We had an idea to challenge another group in trying to track the opponents robot as fast as possible and then head for a predefined home goal. This would be stimulating and force us to implement the robot more efficiently; compare the AI checkers programs that needed to be optimally programmed to work well. (Ref: AI game tournament in AI ground course)

3 Continuous working process of the robot project

3.1 Introduction phase

In our first experiments with the robot at the first lab occasion we closely studied what was possible to do with the robot and its possible potential through the available sensors.

We followed the Kephera user manual closely in order to not break anything and to learn the proper way to handle the robot. The focus of our interest were:

1. The physics of the robot – identified the sensors, LED indicators switches and the different parts of the robot. (Vision turtle, wheels, COM unit)
2. The serial communication protocol – to see the functionalities we were able to access through the serial cable connected to the robot.
3. LabVIEW – We familiarized us with the LabVIEW environment that enabled us to use the communication protocol through a graphic interface that could visualize the sensor values of the robot. We could then through experimentation and with the help of information in the manual grasp the following concepts:
   - Steering system of the robot
   - Speed capabilities of the robot
   - The accuracy of the wheels
   - The accuracy and function of the wheel counters
   - Capacity and values of the 8 infrared sensors (proximity sensors)
   - Visual turrets - visual graphical representation and sensitivity
   - Braitenberg control structure - a method of robot control in an efficient smooth way
4. ROBOWORLD – A simulation tool for the Kephera robot and its environment.

3.2 Planning phase

After achieving an understanding of the basics of the robot we synced our impressions of what was possible within the given frame of time. We realised that Roboworld wouldn't be as useful as we first had thought. The accuracy of Roboworld was good enough for basic understanding, but not for real-time programming feedback. We came to these conclusions:

To use a reactive robot paradigm - since we found that this would be easier to implement than a fully integrated system. To focus on the base goal and the main goal, meaning that we would focus on the navigation part and the graphical representation of the map. Then we moved on to the "follow light behaviour" once the base goal was cleared. We decided to not make any special coding in regard to the extended goal since this would be bad house holding with the limited time we had. Once the first parts were ready we would bother about the extended goal later.

Our agreed agenda thus became:

A. To focus on the base task.
B. Complete the goal task.
C. See how much time was left for extras.

1. Use non lab time for read up on theory and come up with ideas to solve the implementation problems.

2. To use the lab time as effectively as possible through parallel programming and testing at the lab

3. To use a reactive model consisting of following modules:

    - Collision avoidance behaviour
    - Wall following behaviour
    - Map drawing behaviour
    - Localisation behaviour
    - Light source behaviour

Since the base task required for us to graphically visualize a map of the environment after the robot had explored the environment, we discussed different methods to do so.
The robot needed to be able to explore an entire labyrinth no matter how it looked and also needed to be able to avoid objects in it's path. Therefore we came up with a plan of heuristics that the robot could follow regardless of what kind of labyrinth it was put into.

Heuristic plan with add-on modules

1. **Collision avoidance behaviour** – that enabled the robot to freely walk around and register data about the environment without being physically stopped by objects or walls.
   Potential : Draw a map randomly, but may go into same areas over and over again.

2. **Follow wall behaviour** – which made the robot tune its sensors according to the walls and establish a smooth parallel movement alongside the wall it follows.
   Potential : This add-on will make the robot explore all labyrinths without "islands".

3. **Memorize and follow non-visited objects** – When the robot notices other positions that clearly contains a object it memorizes it in a list so that it can go to this area and start a new wall follow behaviour. It will recursively follow all the remaining islands and make a complete exploration of the labyrinth.
   Potential : All labyrinths including any number of "islands" will be explored by the robot.

The algorithm strategy we needed to use would be:

1. Walk and follow wall.
2. Register continuously and draw the visual map.
3. Register areas with a wall or object we don't follow (islands) in a queue.
4. Identify current walls starting point and stop.
5. Remove the positions that has been explored already from the queue.
6 Go to the remaining areas in the queue and restart the algorithm.

This heuristic plan and algorithm would recursively explore any labyrinth.

3.4 Implementation phase

Collision avoidance behaviour

First we had to start with the module that would give the robot a collision avoidance behaviour. Here we simply checked at which proximity sensor values the robot ought to turn and at what speed, so it could move along without hitting any objects or walls that may exist in the environment. This was done quite easily and we got a grasp of the sensitivity of the proximity sensors and the values the walls would reflect from the infrared sensors. The robot now was able to wander around randomly in the environment.

Wall following behaviour

Later we wanted to implement the follow wall behaviour.
We let the left 90 degree sensor adjust itself towards a specific value that would make the robot move closely to the wall. Here we encountered our first problem (P1). The robot would

have an uneven moving pattern and would jerk while it moved around. This was a double problem since each jerk made the wheel counters loose accuracy and it was not nice to watch. The cause was that we set the speed to 0 and changed the angle of the robot in each small turn. We solved this by using a similar method as the Breitenberg cycle to be able to smoothly walk along the walls. By adjusting the speed and turns gradually according to the values it had from the proximity sensors (fig.2) we could finally get a smooth and accurate behaviour that would follow the walls at about 5 mm to 10 mm of distance.

In the wall following we had a few special cases to consider. One was when the wall suddenly disappears and no more values can be read by the left 90 degree proximity sensor.
(a convex turn). We solved this by increasing the right wheel speed until the left 45 degree sensors could spot the wall again.

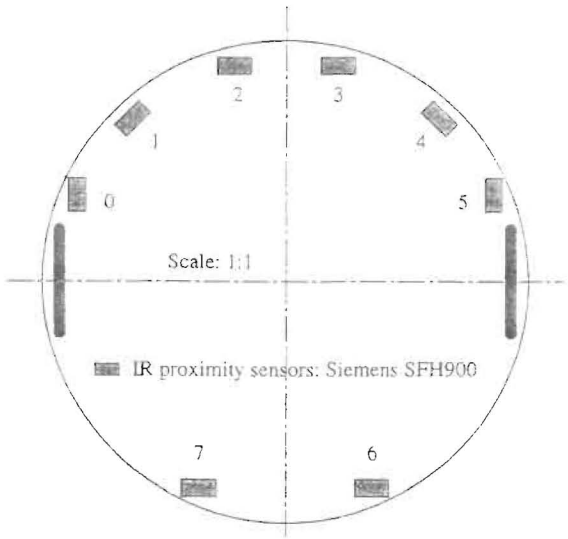


Fig.1:The 8 proximity sensors

Now we encountered another problem (P2). The robot wouldn't move into tight areas and would miss to turn. We adjusted the front sensors so that they would not be as sensitive and by that allowing the robot to turn into tight areas as well. It also encountered problems to move into "v-shaped" walls. We tried to adjust the sensors the same way as we did with the tight turn problem above, but this would give problems in other parts of the robots total behaviour. Therefore we decided to not let the environment consist of any "v-shaped" angles. (Constraint 1)

Visual map representation

Now when the wall following behaviour worked we wanted to focus on the visual representation. Some tests were conducted to see whether the death reckoning strategy based on the wheel counters and the angle of the robot would suffice to keep track of where the robot was at any point. We believed that this together with the proximity sensors to correct for eventual deviations would be accurate enough. We decided to first represent it as basic as possible and therefore chose ASCII symbols in a grid of 100x100 to represent the physical world. Each grid square would be 2x2 cm.

We chose the following map legends to represent different areas in the map:

"-1" - Unknown ground which means the whole map grid consisted of –1 in the beginning.
" 0" - Free ground, where the robot know there are free positions in the map.

" 1" - Where the current wall is positioned on the map.

Then our plan was to let each new object or wall (a potential island) have " 2", " 3", " 4"..., and so on. (Future work)

After the basic map was completed we decided to use JAVA graphics to represent the map and an interactive interface for controlling the robot more easily. We found it more user friendly and also had options to present maps with higher resolutions.

Angle calculation

Now the next step was to make sure that we knew the angle of the robot compared to its start angle. Here we had to refresh our old trigonometric knowledge in order to come up with a correct formula. We aimed to come up with a totally correct formula for any movements made by the robot and got a solution that was a bit too CPU demanding (P3). We decided to approximate some distances to be straight and through this avoid a huge number of calculations that would give almost the same result as the approximation(Constraint 2).

Since it also would depend on the robots ability to not spin the wheels too much at some point relative to its countermovement, and we were unsure of this factor, we decided to go for the approximation. The formula incremented the angle continuously and could give us the robots momentum angle at any time. We calculated the distance the robot travelled at each movement by taking the average incremented value of left and right wheel distance and approximate that the robot went straight forward.

The momentum angle, a of the robot was calculated by subtracting the total distance from left wheel (d_v) with the total distance of the right wheel (d_r) and see the proportion this total distance constitutes of a robot with the diameter of 55mm that performs a full 360 degree turn.

Thus the formula for a:

$a = 360 * (d_v - d_r) / (55 * 2\pi)$

variables:

d_v = total distance on left wheel
d_r = total distance on right wheel
55 = radius, R, on the big circle and diameter on the robot
$R2\pi$= Circumference of a circle with radius R

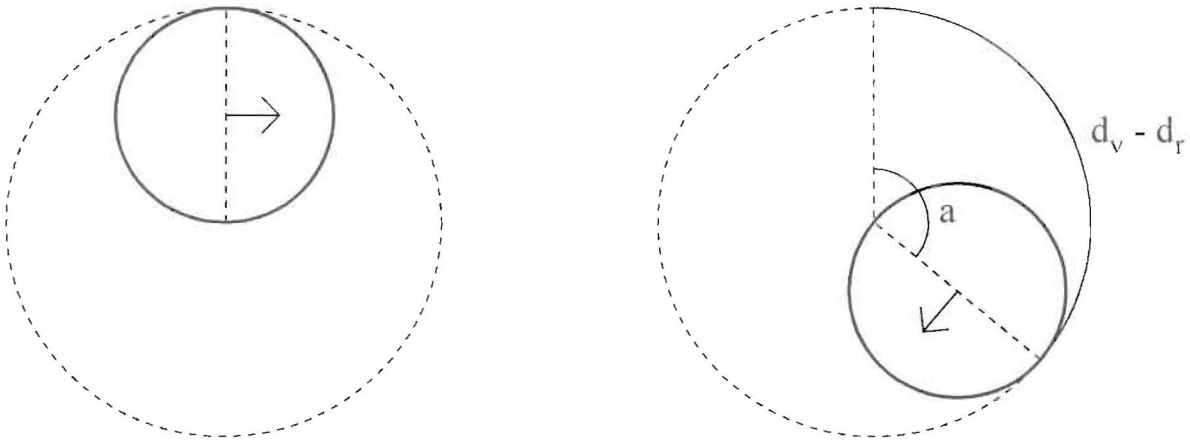


Fig. 2: Angle calculation

We needed to analyse whether the map representation and appreciative calculations were sufficiently corresponding to reality and therefore conducted some validity tests.
We let the robot explore a map and try to stop at the starting position and came to the conclusion that it would find home safely if home was defined ± 4 cm, which means 2 squares in our grid. Here we encountered another problem (P4). The image of the map would be mirrored in the representation. We adjusted the value definitions so that it would correctly display the map as it appeared in reality.

Homing behaviour

Since the robot was also supposed to be able to find a position in the map and go to this point we needed a fairly good accuracy. We made a complex formula for this kind of behaviour that would have to foresee that eventual objects were blocking the robot to move along a straight line from point A to point B anywhere, anytime in a random labyrinth. After some discussions we all agreed that the limited time forced us to postpone this perfect A to B behaviour and transform it into a simpler home position recognizing behaviour that our present robot would manage.

What strategy were we supposed to use? We noticed that the other group were experiencing problems with trying to calculate average proximity values in each sector. This was one of the ideas we had along with some kind of visual turret scans of the environment with built in landmarks. But with little time left we decided to go for a more realistic and less time-consuming approach where we recorded the complete series of convex and concave corners that exist in a random maze in a string. We then lifted the robot and put it back at a random spot in the labyrinth. The robot then followed walls again, but this time only recorded the series of concave and convex corners. This new string it matched against the old string which represented the original map set-up. When the comparation is done bit by bit until the robot knows how many corners it needs to move in order to arrive at the original home position.

We encountered a problem (P5) with identifying these concave and convex corners consequently. To fix this we had to tune the identification of corners better and not build too narrow labyrinths (Constraint 3). The second problem (P6) was that if the labyrinth consisted of a symmetric set-up of concave and convex corners the matching would fail. By avoiding

these labyrinth we escaped the problem.(Constraint 4) We wouldn't have needed to make the robot move around the full labyrinth since only a few corners would have been enough to settle where the robot was relatively to the original map set-up. For total accuracy in finding home we decided to let it complete a full lap.

Light following behaviour

When the base goal was completed we had to go on with the main goal. Which was to make the robot follow a light source. For practical reasons we used a big flashlight and therefore could freely move it around in different sections of the labyrinth. This behaviour was very similar to the Braitenberg method integrated in the wall following method. We just added the light from the flashlight as another factor the robot had to adjust itself to.

When there was no light it would only use wall following behaviour and when the light appeared it would move towards it until the light was so strong that it would consider itself to have found the light source. If the light then was abruptly removed it would just switch back to the wall following behaviour once again. The only problem (P7) we encountered in this section was that if the light happened to project onto the walls there may be a conflict between the lit up wall and some part of the lit up labyrinth ground. The robot would then stand in some kind of deadlock and not know where to move. If only the walls were lit up it would just continue straight on into the wall since it saw the wall as some kind of light it needs to follow. We solved this by deciding that light should only come from straight above. (Constraint 5) After adding this constraint everything worked fine and the robot would smoothly move in the labyrinth until it would track some light source and move towards it.

4 Problems, solutions and constraints

P1: Jerky moving pattern causing inaccuracy
Cause: Speed of wheels and turns of the robot no synchronized.
Solution: Adjusting speed and turns gradually and simultaneously , inspired by Braitenberg cycle. (Well known algorithm for control of robot)

P2: Turning problems in tight labyrinths and v-shaped walls.
Cause: The front sensors were not set correctly according to what was needed.
Solution: To use all of the front sensors and adjust them accordingly.
Constraint1: Avoid environments with v-shaped walls.

P3: Exact calculation of angle and distance was too CPU demanding and not needed since the robots death reckoning deviation after further testing was expected too high.
Cause: The robot will make numerous of small turns and movements and each movement will demand calculations. Each turn will give some small deviation in the death reckoning.
Constraint2: Approximate turns to be straight and use the average distance value of the two wheels.

P4: Graphical representation mirrored compared to real world.
Cause: Implementational mismatch
Solution: Correct the values so that they were displayed to show the real world.

P5: Identifying concave and convex corners
Cause: The sensors were not correctly tuned to adequately identify the corner types.
Solution: Tuning them more adequately and use all relevant sensors in the process.
Constraint3 : Not have too narrow convex and concave corners in the environment.

P6: Identifying corners in symmetric labyrinths
Cause: The list of convex and concave corners may match another labyrinths list.
Solution: Integrate current angle in the corner list (future work)
Constraint4: Avoid symmetric labyrinths and symmetric corner list set-ups.

P7 Conflict between light source and light on a wall
Cause: The sensor values of a wall with a light on, would match the values of the light.
Constraint5: Only use a light source that come from straight above the environment.

5 Results and the final status of the robot

The status of the robot right now :

Base goal

- Collision avoidance behaviour - Works generally well and is included in the wall following behaviour. But in some very narrow environments the robot will manoeuvre with a tight marginal only.
- Wall following behaviour - Works well unless the walls are v-shaped and labyrinth is too tight.
- Map drawing behaviour - Draws a correct map in JAVA graphic representing the environment.
- Localisation behaviour - Robot may be lifted and then finds home position without, problems unless the labyrinth and corner set-up is symmetric.

Main goal

- Light source behaviour - Finds a light source and follows wall, alternatively without problems.

Additional goal

Robot game - Finds an opponent robot with a light source projected on top

of it. However not tested in reality since we lack an autonomous robot opponent. Homing behaviour may be initialised by a click in interface or automatically (future work) as soon as it finds the opponent.

6 Final Analysis

We find that the reactive model was a good choice since we could solve problems in an "ad hoc" way. By dividing the complete task in different behaviours the work was simplified and less complex. The divide and conquer strategy for the complete project was therefore a successful one. If we would have done this project over again we would probably use the reactive model once again. Knowing then that it is quite difficult to match the robot world with the real world due to uneven light conditions, uneven sensor readings, death reckoning deviations, real time programming bugs and our own inexperience in robot programming, we would set aside more work hours for the navigation part of the project. We also found that it is very important to plan each step ahead so that the time is used as effectively possible. We would probably have avoided the fact that we didn't have any practical use of the graphical representation of our robot at all, if we had discussed this in the planning phase.

7 Future work on the robot

We would want to implement the following in an improved version of our robot.

- A complete heuristic strategy so that the robot may explore all possible labyrinths (including islands)
- A test of the generated map to see if the map contains a symmetric sequence of convex and concave corners.
- To implement a follow wall routine with right walls so that it may find the fastest route back to home position.
- To implement a "walk from A to B behaviour".
- To add the measurement of the lengths of each wall so that we can have more information about the environment in order to record more unique data .

8 Opinions

Overall we found that the project was very interesting and stimulating. But in order to achieve better results we wish to have had more time so that we could have planned each phase better. The robot was adequate for its purpose to give us a good, practical learning environment. Maybe we wished for better accuracy of the sensors in some moments but this may be partly due to our own way of implementing the robot. This course made us all very inspired and we would probably be interested in working with robots again. The lectures and seminars have helped us a great deal to manage our results.

9 References

Lectures and seminars
AI Robot course 125, HT-00, Jacek Malec, 2000

Manuals
Kephera Manual , K-Team, 1998

Literature
Mobile Robotics: a practical introduction, Chapter 5, Navigation
Artificial Intelligence : A modern approach, S.Russel, P.Norvig, 1995

Seminar papers
Intelligence without reason, Rodney A. Brooks, 1991

Examination reports
"Object oriented robot control and real time simulation" , Tim Portnoff, 1999
"Object oriented robot simulator", Henrik Lernmark, 1999

Software
RoboWorld, Henrik Lermark, Tim Portnoff, 1999

Appendix A – Java program

```
class AI {

    public static void main(String[] args) {
        Window w = new Window("AI for Robots");
    }

}


class Window extends Frame implements ActionListener{

    Table table;
    Button quitButton;
    Button collisionAvoidButton;
    Button followLightOrWall;
    Button mapNPos;
    Button localize;
    Panel panel;
    Robot kid;
    static int          mapWidth=100,mapHeight=100;


    public Window(String s) {
        super(s);
        /************************************/
        /*    Fönster hantering stuff       */
        /************************************/
        table = new Table();
        quitButton = new Button("Quit");
        collisionAvoidButton = new Button("Collision Avoidance");
        followLightOrWall = new Button("Follow Light");
        mapNPos = new Button("Create Map");
        localize = new Button("Find Home");
        panel = new Panel();
        this.setBounds(100, 100, 500, 500);
        table.setSize(new Dimension(400, 400));
        this.add("Center", table);
        panel.add("Center", quitButton);
        panel.add("Center", collisionAvoidButton);
        panel.add("Center", followLightOrWall);
        panel.add("Center", mapNPos);
        panel.add("Center", localize);
        this.add("South", panel);
        this.pack();
        this.show();
        quitButton.addActionListener(this);
        collisionAvoidButton.addActionListener(this);
        followLightOrWall.addActionListener(this);
        mapNPos.addActionListener(this);
        localize.addActionListener(this);
        /************************************/
        /*    SLUT fönster hantering stuff  */
        /************************************/

        int speedLeft,speedRight;
        kid = new Robot(mapWidth,mapHeight);
        kid.setupSerialConnection();

    }
    /************************************/
    /*   SJÄLVA KODEN                   */
    /************************************/

    /*Collision Avoidiance*/
```

```
    public void CA() {
        boolean quit = false;
        while (true) {
            kid.Braitenberg();
            kid.testZero();

        }
    }

    /*Se Robot.followWall2Pos*/
    public void mapAndPos() {
        int x,y;

//      infile keyb = new infile();
//      if (!keyb.Open()) {
//          System.out.println("Kan ej öppna tangentbordet.");
//          System.exit(0);
//      }

        kid = new Robot(mapWidth,mapHeight);
        //kid.setupSerialConnection();
        kid.setMapMaking(true);
        kid.goToCorner();
        kid.reset();
        System.out.println("#############START!!!#################");
        kid.followWall2Pos(mapWidth/2,mapHeight/2);
        table.setMap(kid.getModifiedMap());
        table.repaint();
        kid.setMapMaking(false);
    }

    /*Följer ljus, men om det inte registrerar
      något ljus, så följer den väggen.*/
    public void followLightOrWall(){
        int[] sens;
        boolean seeLight=false;
        boolean sawLight=true;

        while(true) {
            sens = kid.getLightSensors();
            for (int i=0;i<sens.length-2;i++)
                if (sens[i]<450)
                    seeLight = true;
            if (seeLight)
                kid.go2theLight(sens);
            else {
                //eg. sawLight!
                if (sawLight) {
                    kid.untilFoundWall();
                    kid.adjust2Wall();
                }
                else
                    kid.followWall(kid.getProxSensors());
            }
            sawLight = seeLight;
            seeLight=false;
        }
    }

    /*Se Robot.localize*/
    public void localize() {
        kid.localize();
    }

    /*Vad som ska göras*/
    public void actionPerformed(ActionEvent e) {
        Object object = e.getSource();

        if (object == quitButton) {
            kid.turnLeft(kid.map.getAngle());
            kid.emergencyBrake();
            System.exit(0);
        }
        else if (object == collisionAvoidButton) {
            CA();
        }
        else if (object == followLightOrWall) {
            followLightOrWall();
        }
        else if (object == mapNPos) {
            mapAndPos();
        }
```

```
        else if (object == localize) {
            localize();
        }
        else
            System.out.println("ERROR!!!");

    }

}

/*************************************/
/*   END: SJÄLVA KODEN               */
/*************************************/

class Table extends Canvas {

    //  private Graphics graphics;
    private int dx, dy;
    private int x, y;
    private int height, width;
    public int[][] map;

    public Table() {
        dx = 4;
        dy = 4;
        x = 0;
        y = 0;
        height = 100;
        width = 100;
    }

    public void setCoords(int x, int y) {
        this.x = x;
        this.y = y;
    }

    public void setMap(int[][] map) {
        this.map = map;
    }

    public void update(Graphics g) {
        g.setColor(Color.white);
        for (int i = height - 1; i >= 0; i--)
            for (int j = 0; j < width; j++)
                if (map[j][i] == 0) {
                    setCoords(j, i);
                    g.fillRect(x*dx,400-y*dy,dx,dy);
                }
    }

}


class Robot{
    private static final int LF = 10; // LineFeed avslutar alla Khepera-reply

    private static final int DIA = 55;  //Khepera's diameter i mm
    private static final double MOVEUNIT = 0.08; //Enhet som Khepera's hjul använder, i mm
    private static final double OMKRETS = Math.PI*DIA; //Khepera's omkrets i mm
    private static final int NufUnitsInCirc = (int)(OMKRETS/MOVEUNIT);  //antal hela
"hjulenheter" som går i omkretsen

    /*Used when checking robot mode */
    private static final int T_MOVING=0;
    private static final int T_ON_TARGET=1;
    private static final int M_SPEED_MODE=0;
    private static final int M_POS_MODE=1;
    private static final int M_PWM_MODE=0;

    /*sensors*/
    static final int SENS_L90=0;
    static final int SENS_L45=1;
    static final int SENS_L10=2;
    static final int SENS_R10=3;
    static final int SENS_R45=4;
    static final int SENS_R90=5;
    static final int SENS_BR=6;
    static final int SENS_BL=7;

    int speedLeft=0,speedRight=0;
    int nufZero = 0;
```

```
int cruiseSpeed = 10;
boolean makeMap;
//int positionLeft=0,positionRight=0;

static OutputStream outputStream;
static InputStream inputStream;

Map map;
String originalCorners;
int mapWidth,mapHeight;
int[] saved_robotPos;

/*20 hörn och konvex/konkav,XCoord,YCoord,angle per hörn*/
/*0 för konkav(högersväng) och 1 för konvex(vänstersväng)*/
int[][] cornerMatrix = new int[20][4];
int nufCorners;
static final int matrix_CORNER = 0;
static final int matrix_X = 1;
static final int matrix_Y = 2;
static final int matrix_ANGLE = 3;
static final int KONKAV = 0;
static final int KONVEX = 1;

/*Sensor limits for the corners*/
static final int KONVEX_LIMIT=1;
static final int KONKAV_LIMIT=1000;

/*Konstuktorn*/
public Robot(int width,int height) {
    mapWidth = width;
    mapHeight = height;
    map = new Map(mapWidth,mapHeight);
    originalCorners = "";
}

/*****************************************************/
/*****************************************************/
/*                    BEHAVIORS                      */
/*****************************************************/
/*****************************************************/

/*Gå framåt tills något hinder hittas*/
public void untilFoundWall() {
    int CLOSENESS = 1000;
    int[] sens;
    setSpeed(cruiseSpeed,cruiseSpeed);
    boolean foundWall = false;
    while (!foundWall) {
        sens = getProxSensors();
        for (int i=0;i<sens.length;i++) {
            if (sens[i]>CLOSENESS) {
                foundWall = true;
                break;
            }
        }
    }
}

/*Vrid roboten så att dess vänstra
  sida är parallell mot väggen.
  Förutsätter att roboten befinner sig
  nära en vägg*/
public void adjust2Wall() {
    int[] sens;
    int i;
    setSpeed(cruiseSpeed,-cruiseSpeed);
    boolean adjusted = false;
    while (!adjusted) {
        adjusted = true;
        sens = getProxSensors();
        for (i=1;i<sens.length;i++) {
            if (sens[SENS_L90]<sens[i]/*+300*/) {
                adjusted = false;
            }
        }
    }
    setSpeed(0,0);
}

/*Vår reaktiva collision avoidiance behavior
  (influerad av Braitenberg)*/
public void Braitenberg() {
```

```
    float DEGREE_90 = -2f;
    float DEGREE_45 = -4f;
    float DEGREE_10 = -6f;
    int[] sens = getProxSensors();
    float sideLeft = sens[SENS_L90]*(DEGREE_90/1023f);
    float diagLeft = sens[SENS_L45]*(DEGREE_45/1023f);
    float frontLeft = sens[SENS_L10]*(DEGREE_10/1023f);

    float frontRight = sens[SENS_R10]*(DEGREE_10/1023f);
    float diagRight = sens[SENS_R45]*(DEGREE_45/1023f);
    float sideRight = sens[SENS_R90]*(DEGREE_90/1023f);

    int speedR = (int)(cruiseSpeed+sideLeft+diagLeft+frontLeft);
    int speedL = (int)(cruiseSpeed+sideRight+diagRight+frontRight);

    /*skicka iväg hastigheterna*/
    setSpeed(speedL,speedR);
}

/*Ett reaktivt (vänster)vägg-följar beteende.
  Förutsätter att roboten befinner sig nära en vägg i början.
  Stannar när den nått x,y i (obs)kartans koordinater!
  (origoX=Map.Width/2 och origoy=Map.Height/2)
*/
public void followWall2Pos(int x,int y) {
    int[] sens;
    boolean foundPos=false;
    boolean leftTurn=false,rightTurn=false;
    nufCorners = 0;
    boolean isCorner = false;

    /*Följ vägg*/
    while (!foundPos) {
        sens = getProxSensors();
        map.updatePos(getPos());
        if (!leftTurn) {
            /*Konvex hörn*/
            if (sens[SENS_L45]<KONVEX_LIMIT){
                leftTurn=true;
                rightTurn=false;
                cornerMatrix[nufCorners][matrix_CORNER]=KONVEX;
                cornerMatrix[nufCorners][matrix_X]=map.getX();
                cornerMatrix[nufCorners][matrix_Y]=map.getY();
                cornerMatrix[nufCorners][matrix_ANGLE]=map.getAngle();
                nufCorners++;
                isCorner = true;
            }
            /*Konkav hörn*/
            else if (sens[SENS_L10]>KONKAV_LIMIT) {
                if (!rightTurn) {
                    rightTurn=true;
                    cornerMatrix[nufCorners][matrix_CORNER]=KONKAV;
                    cornerMatrix[nufCorners][matrix_X]=map.getX();
                    cornerMatrix[nufCorners][matrix_Y]=map.getY();
                    cornerMatrix[nufCorners][matrix_ANGLE]=map.getAngle();
                    nufCorners++;
                    isCorner = true;
                }
            }
            /*rakt fram*/
            else {
                rightTurn = false;
            }
        }
        else {
            if ((sens[SENS_L45]>600) && (sens[SENS_L90]>850))
                leftTurn = false;
        }

        /*Map making*/
        if (makeMap)
            map.set();

        followWall(sens);

        /*kollar om man nått målet (vid hörn)*/
        if (isCorner) {
            isCorner = false;
            if (map.reachedPos(x,y)) {
                emergencyBrake();
                foundPos=true;
            }
```

```
        }
    }

    for (int i=0;i<nufCorners;i++)
        System.out.print(cornerMatrix[i][matrix_CORNER]);
    System.out.println();

    originalCorners = corner_Matrix2String(cornerMatrix);
    System.out.println("originalCorners:  "+originalCorners);
    saveContext();
}

/*Försöker lokalisare sig genom att ställa sig vid starthörnet
 (vid mapmaking). Detta görs genom att titta på alla hörn och
 sedan jämföra dessa hörn med hörnen man hade då man gjorde
 kartan (mapmaking)*/
public void localize(){
    int[] sens;
    boolean foundPos=false;
    boolean leftTurn=false,rightTurn=false;
    String newCorners ="";
    boolean isCorner = false;
    int x = mapWidth/2;
    int y = mapHeight/2;
    goToCorner();
    reset();
    System.out.println("#####START LOCALIZE#####");
    Map localMap = new Map(mapWidth,mapHeight);
    localMap.setMapMaking(true);

    nufCorners=0;
    /*Följ vägg*/
    while (!foundPos) {
        sens = getProxSensors();
        localMap.updatePos(getPos());
        if (!leftTurn) {
            if (sens[SENS_L45]<KONVEX_LIMIT){
                leftTurn=true;
                rightTurn=false;
                cornerMatrix[nufCorners][matrix_CORNER]=KONVEX;
                cornerMatrix[nufCorners][matrix_X]=localMap.getX();
                cornerMatrix[nufCorners][matrix_Y]=localMap.getY();
                cornerMatrix[nufCorners][matrix_ANGLE]=localMap.getAngle();
                nufCorners++;
                isCorner = true;
            }
            else if (sens[SENS_L10]>KONKAV_LIMIT) {
                if (!rightTurn) {
                    rightTurn=true;
                    cornerMatrix[nufCorners][matrix_CORNER]=KONKAV;
                    cornerMatrix[nufCorners][matrix_X]=localMap.getX();
                    cornerMatrix[nufCorners][matrix_Y]=localMap.getY();
                    cornerMatrix[nufCorners][matrix_ANGLE]=localMap.getAngle();
                    nufCorners++;
                    isCorner = true;
                }
            }
            else {
                rightTurn = false;
            }
        }
        else
            if ((sens[SENS_L45]>600) && (sens[SENS_L90]>850))
                leftTurn = false;


        followWall(sens);

        /*kollar om man nått målet (vid hörn)*/
        if (isCorner) {
            isCorner = false;
            if (localMap.reachedPos(x,y)) {
                emergencyBrake();
                foundPos=true;
            }
        }
    }
    for (int i=0;i<nufCorners;i++)
        System.out.print(cornerMatrix[i][matrix_CORNER]);
    System.out.println();

    newCorners = corner_Matrix2String(cornerMatrix);
```

```
    System.out.println("Original hörn: "+originalCorners);
    System.out.println("Dessa hörn: "+newCorners);

    int cornerNr;
    cornerNr = compareCorners(originalCorners,newCorners);
    if (cornerNr!=-1) {
        System.out.println("Gå "+cornerNr+"hörn framåt!");
        go2CornerNr(cornerNr);
        emergencyBrake();
        restoreContext();
    }
}


/*Förflyttar sig corners hörn fram.
  Förutsätter att man befinner sig vid(precis före)
  ett konkavt hörn*/
public void go2CornerNr(int corners){
    int[] sens;
    boolean reachedCorner=false;
    boolean leftTurn=false,rightTurn=false;
    Map localMap = new Map(mapWidth,mapHeight);
    int x=-1,y=-1;
    int newX,newY;
    reset();
    /*lägger till 1 för att kompensera första hörnet som ej räknas!*/
    if (corners>0)
        corners++;

    /*Följ vägg*/
    while (corners!=0) {
        sens = getProxSensors();
        if (!leftTurn) {
            if (sens[SENS_L45]<KONVEX_LIMIT){
                leftTurn=true;
                rightTurn=false;
               newX=localMap.getX();
               newY=localMap.getY();
               System.out.println("X:"+newX+"\tY:"+newY);
               /*Ej dubletter om:*/
               if ((newX > x) || (newX < x ) ||
                   (newY > y) || (newY < y)) {
                   corners--;
                   x = newX;
                   y = newY;
               }
               else
                   System.out.println("DUBLETT!");
            }
            else if (sens[SENS_L10]>KONKAV_LIMIT) {
                if (!rightTurn) {
                    rightTurn=true;
                   newX=localMap.getX();
                   newY=localMap.getY();
                   System.out.println("X:"+newX+"\tY:"+newY);
                   /*Ej dubletter om:*/
                   if ((newX > x) || (newX < x ) ||
                       (newY > y) || (newY < y)) {
                       corners--;
                       x = newX;
                       y = newY;
                   }
                   else
                       System.out.println("DUBLETT!");
                }
            }
            else {
                rightTurn = false;
            }
        }
        else
            if ((sens[SENS_L45]>600) && (sens[SENS_L90]>850))
                leftTurn = false;
        localMap.updatePos(getPos());
        followWall(sens);
    }
    emergencyBrake();
}

/*Ställer sig vid första bästa (eg. andra) konkava hörn*/
public void goToCorner() {
    int[] sens;
```

```
        int corners=0;
        boolean leftTurn=false,rightTurn=false;
        untilFoundWall();
        adjust2Wall();

        /*Följ vägg*/
        while (corners<2) {
            sens = getProxSensors();
            if (!leftTurn) {
                if (sens[SENS_L45]<KONVEX_LIMIT){
                    leftTurn=true;
                    rightTurn=false;
                }
                else if (sens[SENS_L10]>KONKAV_LIMIT) {
                    if (!rightTurn) {
                        rightTurn=true;
                        corners++;
                    }
                }
                else {
                    rightTurn = false;
                }
            }
            else
                if ((sens[SENS_L45]>600) && (sens[SENS_L90]>850))
                    leftTurn = false;
            followWall(sens);
        }
//      emergencyBrake();
    }

        public void followWall(int[] sens) {
        int speed,speedL,speedR;
        float sideLeft,diagLeft,frontLeft;
//      int[] sens;
        boolean foundPos=false;

        /*Följ vägg*/
//      sens = getProxSensors();
        frontLeft = (sens[SENS_L10])*(-8f/1023f);
        sideLeft = (sens[SENS_L90]-500)*(-2f/500f);
        diagLeft = (sens[SENS_L45]-800)*(-5f/500f);//minska 500?Ta bort?
        speed = (int)(frontLeft+sideLeft+diagLeft);

        /*Sväng inåt (minus på vänster hjul)*/
        if (speed>0) {
            speedL = cruiseSpeed-speed;
            speedR = cruiseSpeed;
        }
        /*Sväng utåt (minus på höger hjul)*/
        else {
            speedL = cruiseSpeed;
            speedR = cruiseSpeed+speed;   //speed<0
        }
        setSpeed(speedL,speedR);
    }

    public void go2theLight(int[] sens){
        int speed,speedL,speedR;
        float sideLeft,diagLeft,frontLeft;
        float sideR,diagR,frontR;
        int TOOCLOSE =50;

        frontLeft = (500-sens[SENS_L10])*(-2f/500f);
        diagLeft =  (500-sens[SENS_L45])*(-8f/500f);
        sideLeft =  (500-sens[SENS_L90])*(-10f/500f);
        speedL = cruiseSpeed+(int)(frontLeft+sideLeft+diagLeft);

        frontR = (500-sens[SENS_R10])*(-2f/500f);
        diagR =  (500-sens[SENS_R45])*(-8f/500f);
        sideR =  (500-sens[SENS_R90])*(-10f/500f);
        speedR = cruiseSpeed+(int)(frontR+sideR+diagR);

//      if ((sens[SENS_L10]<TOOCLOSE) || (sens[SENS_L45]<TOOCLOSE) || (sens[SENS_L90]<TOOCLOSE)
||
//          (sens[SENS_R10]<TOOCLOSE) || (sens[SENS_R45]<TOOCLOSE) ||
(sens[SENS_R90]<TOOCLOSE))
//          foundLight=true;

        setSpeed(speedL,speedR);
    }
```

```
    /*************************************************/
    /*************************************************/
    /*             END: BEHAVIORS                    */
    /*************************************************/
    /*************************************************/

    /*******************************************************/
    /*******************************************************/
    /*       Diverse hjälpmetoder                          */
    /*******************************************************/
    /*******************************************************/

    /*Sparar robotens stateinfo, dvs dess hjulpositioner,
     */
    public void saveContext() {
        saved_robotPos = getPos();
        map.saveContext();
    }

    public void restoreContext() {
        setPosition(saved_robotPos[0],saved_robotPos[1]);
        map.restoreContext();
    }

    /*Analyserar matrisen med hörn och skapar en sträng
      med de hörn vi "tror" faktiskt finns.
      (Antar att sista och näst sista hörnet i matrisen
      ej är kopior.)*/
    public String corner_Matrix2String(int[][] m) {
        int LIMIT = 0;
        String retCorn="";

        System.out.println("CORNER\t"+"X value\t"+"Y value");
        for (int k=0;k<nufCorners;k++) {
            System.out.println(m[k][matrix_CORNER]+"\t"+m[k][matrix_X]+"\t"+m[k][matrix_Y]);
        }

        for (int i=1;i<nufCorners-1;i++) {
            /*Om det är olika sorts hörn eller
              om (i+1) ligger utanför i:s radie (2),så är de ej dubletter*/
            if ((m[i+1][matrix_CORNER]!=m[i][matrix_CORNER]) ||
                ((m[i+1][matrix_X]>m[i][matrix_X]+LIMIT) || (m[i+1][matrix_X]<m[i][matrix_X]-
LIMIT)) ||
                 ((m[i+1][matrix_Y]>m[i][matrix_Y]+LIMIT) || (m[i+1][matrix_Y]<m[i][matrix_Y]-
LIMIT))))
                if (m[i][matrix_CORNER]==KONKAV)
                    retCorn+="O";
                else if (m[i][matrix_CORNER]==KONVEX)
                    retCorn+="X";
                else
                    System.out.println("Konstiga hörn!");
            else
                System.out.println("Tog bort en dublett. Hörn Nr: "+i);
        }

        /*Kollar första och sista hörnen*/
        /*Hörnen är ej dubletter*/
        LIMIT = 2;
        if ((m[nufCorners-1][matrix_CORNER]!=m[0][matrix_CORNER]) ||
            (((m[nufCorners-1][matrix_X]>m[0][matrix_X]+LIMIT) || (m[nufCorners-
1][matrix_X]<m[0][matrix_X]-LIMIT)) ||
             ((m[nufCorners-1][matrix_Y]>m[0][matrix_Y]+LIMIT) || (m[nufCorners-
1][matrix_Y]<m[0][matrix_Y]-LIMIT)))) {

            System.out.println("Första och sista hörn är EJ dubletter!!");
            if (m[nufCorners-1][matrix_CORNER]==KONKAV)
                retCorn+="O";
            else if (m[nufCorners-1][matrix_CORNER]==KONVEX)
                retCorn+="X";
            else
                System.out.println("Konstiga hörn!");

            if (m[0][matrix_CORNER]==KONKAV)
                retCorn+="O";
            else if (m[0][matrix_CORNER]==KONVEX)
                retCorn+="X";
            else
                System.out.println("Konstiga hörn!");
        }

        /*Dubletter*/
        else {
```

```
            if (m[0][matrix_CORNER]==KONKAV)
                retCorn+="O";
            else if (m[0][matrix_CORNER]==KONVEX)
                retCorn+="X";
            else
                System.out.println("Konstiga hörn!");
        }
        return retCorn;
    }

    /*returnerar hur många steg new ska "cirkulärskiftas"
        åt vänster för att se ut som orig*/
    public int compareCorners(String orig,String newCorn) {
        int difference=0;
        if (orig.length() != newCorn.length()) {
            System.out.println("OLIKA ANTAL HÖRN!!!");
            difference=-1;
            //System.exit(1);
        }
        else
            while (orig.compareTo(newCorn)!=0) {
                newCorn = newCorn.substring(1)+newCorn.charAt(0);
                difference++;
            }
        return difference;
    }

    public int[][] getMap() {
        return map.getMap();
    }

    public int[][] getModifiedMap() {
        return map.getModifiedMap();
    }

    public void setMapMaking(boolean b) {
        makeMap = b;
        map.setMapMaking(b);
    }

    /*Kollar om roboten fastnat i en återvändsgränd
      i så fall vänd 180 grader*/
    public void testZero() {
        if ((speedLeft==0)&&(speedRight==0)) {
            nufZero++;
            if (nufZero>5) {
                emergencyBrake();
                turnLeft(180);
            }
        }
        else
            nufZero=0;
    }
    /*********************************************************/
    /*********************************************************/
    /*       END: Diverse hjälpmetoder                       */
    /*********************************************************/
    /*********************************************************/



    /*************************************************/
    /*************************************************/
    /*         METHODS FOR CONTROLLING KHEPERA       */
    /*************************************************/
    /*************************************************/

    public boolean reachedPos() {
        int[] motionStat = getMotionStatus();
        return ((motionStat[0] == T_ON_TARGET) && (motionStat[3] == T_ON_TARGET));
    }

    public void reset() {
//      setSpeed(0,0);
        setPosition(0,0);
    }

    public int[] getMotionStatus(){
        sendMessage("K\n");
        return getIntReply("k");
    }
```

```
public void setSpeed(int leftWheel,int rightWheel) {
    sendMessage("D,"+leftWheel+","+rightWheel+"\n");
    getReply();
    speedLeft = leftWheel;
    speedRight = rightWheel;
}

public void setPosition(int leftWheel,int rightWheel){
    sendMessage("G,"+leftWheel+","+rightWheel+"\n");
    getReply();
    //      postionLeft = leftWheel;
    //      positionRight = rightWheel;

}

public void setReachPosition(int leftWheel,int rightWheel){
    sendMessage("C,"+leftWheel+","+rightWheel+"\n");
    getReply();
    //      postionLeft = leftWheel;
    //      positionRight = rightWheel;
}

public void emergencyBrake() {
    sendMessage("D,0,0\n");
    getReply();
}

public int[] getProxSensors() {
    sendMessage("N\n");
    return getIntReply("n");
}

public int[] getPos() {
    sendMessage("H\n");
    return getIntReply("h");
}

public int[] getLightSensors() {
    sendMessage("O\n");
    return getIntReply("o");
}

public int[] getVisTurretSensors() {
    sendMessage("T,2,N\n");
    return getIntReply("t,2,n");
}

/*Robert Pallbo's*/
/** Sends a message to the robot. The reply is returned. Failed
    transmission results in a warning printed on the consol */
public static void sendMessage(String msg){
    try {
        outputStream.write(msg.getBytes());
    } catch (IOException e) {
        System.out.println("Warning: Message could not be sent to robot");
        System.out.println("Message was: " + msg);
        System.out.println(e.getMessage());
    }
}


/*Robert Pallbo's*/
/** Waits until data is available from the Khepera */
public static void waitForReply(){
    try {
        while(inputStream.available()<=0) {}
    } catch (IOException e) {
        System.out.println("Warning: Waiting for reply failed");
        System.out.println(e.getMessage());
    }
}


/*Robert Pallbo's*/
/** Reads a reply or a message from the robot. */
public static String getReply(){
    String reply = "";

    // Läs in data fram till LineFeed
    try {
        int b = inputStream.read();
        while(b!=LF) {
```

```
            reply += (char) b;
            b = inputStream.read();
        }
    } catch (IOException e) {
        System.out.println("Warning: reply from robot could not be read");
        System.out.println(e.getMessage());
    }

    // Returnera resultatet
    return reply;
}

/*Robert Pallbo's*/
/** Reads the reply from Khepera and puts the reply in an int array.
    The prefix in the reply is ignored when parsing the reply. */

public int [] getIntReply(String prefix){

    // Läs in responsen som en sträng och plocka bort prefixet
    String reply = getReply();
    reply = reply.substring(reply.indexOf(prefix)+prefix.length()+1);

    // Använd en tokenizer för att separera substrängarna med värdena
    StringTokenizer token = new StringTokenizer(reply, ",\n\r\f ");
    int result [] = new int [token.countTokens()];

    // Parsa substrängarna med värdena
    for(int i=0; i<result.length; i++){
        String siffror = token.nextToken();
        try {
            result[i] = Integer.parseInt(siffror);
        } catch (Exception e) {
            System.out.println("Warning: Problem parsing values");
            System.out.println("String to parse: " + reply);
            System.out.println(e);
        }
    }

    return result;
}

/*Robert Pallbo's*/
/** Initierar seriekommunikationen */
public static void setupSerialConnection(){
    CommPortIdentifier portId;
    SerialPort serialPort;

    // Öppna /dev/modem
    try {
        portId = CommPortIdentifier.getPortIdentifier("/dev/modem");
        serialPort = (SerialPort) portId.open("KheperaController", 2000);
        outputStream = serialPort.getOutputStream();
        inputStream = serialPort.getInputStream();
        serialPort.setSerialPortParams(19200,
                                       SerialPort.DATABITS_8,
                                       SerialPort.STOPBITS_2,
                                       SerialPort.PARITY_NONE);
    } catch (NoSuchPortException e) {
        System.out.println("No such port");
        System.out.println(e.getMessage());
        System.exit(0);
    } catch (PortInUseException e) {
        System.out.println("The port is already open");
        System.out.println(e.getMessage());
        System.exit(0);
    } catch (IOException e) {
        System.out.println("No output or input stream could be " +
                           "created for the port");
        System.out.println(e.getMessage());
        System.exit(0);
     } catch (UnsupportedCommOperationException e) {
        System.out.println("The serial port could not be configured");
        System.out.println(e.getMessage());
        System.exit(0);
     }

}

public void frontalCollision() {
    /*sensibility*/
    int FLSens = 900;
    int FRSens = 900;
```

```
        int[] sensors = getProxSensors();
        if (sensors[SENS_L10]>FLSens || sensors[SENS_R10]>FRSens) {
            turnLeft(90);   //så länge.
        }
    }

    public void turnLeft(int angle) {
        //spara undan hastigheten
        int oldSpeedL = speedLeft;
        int oldSpeedR = speedRight;
        emergencyBrake(); //vill eg. bara sätta hast=0!!
        //Beräknar hur mycket varje hjul ska snurra
        int turnUnits = (int)(NufUnitsInCirc*angle/360);
        //Snurra hjulen
        setPosition(0,0);
        setReachPosition(-turnUnits,turnUnits);
        //Kontollera om nått positionen?
        while(!reachedPos());  //busy wait!!
        //Sätter roboten till ursprungshast.
        setSpeed(oldSpeedL,oldSpeedR);
    }

    public void turnRight(int angle) {
        //spara undan hastigheten
        int oldSpeedL = speedLeft;
        int oldSpeedR = speedRight;
        emergencyBrake(); //vill eg. bara sätta hast=0!!
        //Beräknar hur mycket varje hjul ska snurra
        int turnUnits = (int)(NufUnitsInCirc*angle/360);
        //Snurra hjulen
        setPosition(0,0);
        setReachPosition(turnUnits,-turnUnits);
        //Kontollera om nått positionen?
        while(!reachedPos());  //busy wait!!
        //Sätter roboten till ursprungshast.
        setSpeed(oldSpeedL,oldSpeedR);
    }

    public boolean senseFront(int frontLeft,int frontRight) {
        int[] sensors = getProxSensors();
        if (sensors[2]>frontLeft || sensors[3]>frontRight)
            return true;
        else
            return false;
    }

    /************************************************/
    /************************************************/
    /*      END: METHODS FOR CONTROLLING KHEPERA    */
    /************************************************/
    /************************************************/


}


    /************************************************/
    /************************************************/
    /*      END:         ROBOT                      */
    /************************************************/
    /************************************************/



class Map{

    int width,height;
    /*-1 okänt; 0 fritt; 1 hinder(används ej)*/
    int[][] map;
    boolean makeMap;

    private static final int DIA = 110;  //2 x Khepera's diameter i mm
    private static final float MOVEUNIT = 0.08f; //Enhet som Khepera's hjul använder, i mm
    private static final float OMKRETS = (float)(Math.PI*DIA); //Omkrets("2 x kheperas") i mm
    private static final float NufUnitsInCirc = OMKRETS/MOVEUNIT;  //antal "hjulenheter" som
går i omkretsen
    private static final float k = 55/(2*MOVEUNIT);   //Halva khepera i MOVEUNITS
    private static final float kDiag = (float)(k*Math.sqrt(2));

    /*Robotens koordinater i MOVEUNIT enheter*/
    float xCoord,yCoord;
```

```
double angle;
int oldLPos,oldRPos;
int origoX,origoY;

/*Undansparad info*/
float saved_xCoord,saved_yCoord;
double saved_angle;
int saved_oldLPos,saved_oldRPos;
int saved_origoX,saved_origoY;

public Map(int x,int y) {
    xCoord=0;
    yCoord=0;
    oldLPos=0;
    oldRPos=0;
    angle=0;
    /*initierar kartan*/
   origoX=x/2;
   origoY=y/2;
    width = x;
    height = y;
    map = new int[width][height];

    for(int i=0;i<width;i++)
        for(int j=0;j<height;j++)
            map[i][j]=-1;
}

public void saveContext() {
    saved_xCoord = xCoord;
    saved_yCoord = yCoord;
    saved_angle  = angle;
    saved_oldLPos= oldLPos;
    saved_oldRPos= oldRPos;
    saved_origoX = origoX;
    saved_origoY = origoY;
}
public void restoreContext() {
    xCoord = saved_xCoord;
    yCoord = saved_yCoord;
    angle  = saved_angle;
    oldLPos= saved_oldLPos;
    oldRPos= saved_oldRPos;
    origoX = saved_origoX;
    origoY = saved_origoY;
}

public void updatePos(int[] pos) {
    float distance;  //distance traveled in MOVEUNIT
    int localL,localR;
    localL=pos[0]-oldLPos;
    localR=pos[1]-oldRPos;
    oldLPos = pos[0];
    oldRPos = pos[1];
    //      distance = Math.min(localL,localR);
    distance = (localL+localR)/2f; //försöka ta minsta värdet istället?
    xCoord = xCoord+(float)(Math.sin(angle)*distance);
    yCoord = yCoord+(float)(Math.cos(angle)*distance);
    //räkna ut ny vinkel (använd gamla först!)
    angle=2*Math.PI*(oldLPos-oldRPos)/NufUnitsInCirc;
}

/*Sätter robotens 1+8 punkter på kartan som ledig*/
/*Är det inte tillräcklig med 1+3 punkter?*/
public void set() {
    float x,y;
    float beta = (float)(Math.PI/4-angle);  //45grd - alpha
    /*Mitten*/
    map[Math.round(xCoord/250)+origoX][Math.round(yCoord/250)+origoY]=0;
    /*Nord(1)*/
    x=(float)(xCoord+k*Math.sin(angle));
    y=(float)(yCoord+k*Math.cos(angle));
    map[Math.round(x/250)+origoX][Math.round(y/250)+origoY]=0;
    /*NordÖst(2)*/
    x=(float)(xCoord+kDiag*Math.cos(beta));
    y=(float)(yCoord+kDiag*Math.sin(beta));
    map[Math.round(x/250)+origoX][Math.round(y/250)+origoY]=0;
    /*Öst(3)*/
    x=(float)(xCoord+k*Math.cos(angle));
    y=(float)(yCoord-k*Math.sin(angle));
    map[Math.round(x/250)+origoX][Math.round(y/250)+origoY]=0;
    /*SydÖst(4)*/
```

```
    x=(float)(xCoord+kDiag*Math.sin(beta));
    y=(float)(yCoord-kDiag*Math.cos(beta));
    map[Math.round(x/250)+origoX][Math.round(y/250)+origoY]=0;
    /*Syd(5)*/
    x=(float)(xCoord-k*Math.sin(angle));
    y=(float)(yCoord-k*Math.cos(angle));
    map[Math.round(x/250)+origoX][Math.round(y/250)+origoY]=0;
    /*SydVast(6)*/
    x=(float)(xCoord-kDiag*Math.cos(beta));
    y=(float)(yCoord-kDiag*Math.sin(beta));
    map[Math.round(x/250)+origoX][Math.round(y/250)+origoY]=0;
    /*Vast(7)*/
    x=(float)(xCoord-k*Math.cos(angle));
    y=(float)(yCoord+k*Math.sin(angle));
    map[Math.round(x/250)+origoX][Math.round(y/250)+origoY]=0;
    /*NordVast(8)*/
    x=(float)(xCoord-kDiag*Math.sin(beta));
    y=(float)(yCoord+kDiag*Math.cos(beta));
    map[Math.round(x/250)+origoX][Math.round(y/250)+origoY]=0;
}

/*returnerar robotens nuvarande vinkel i grader
  (jämfört med initialvinkeln)*/
public int getAngle() {
    return (int)(Math.toDegrees(angle)); //Alt. (int)(Math.rint())
}

//returnerar robotens x koordinat i kartan
public int getX() {
    return Math.round(xCoord/250)+origoX;
}
//returnerar robotens y koordinat i kartan
public int getY() {
    return Math.round(yCoord/250)+origoY;
}

//returnerar robotens x koordinat i "robotenheter"
public float getRealX() {
    return xCoord;
}
//returnerar robotens y koordinat i "robotenheter"
public float getRealY() {
    return yCoord;
}

/*returnerar true om robotenbefinner sig på (x,y)
  i kartkoordinater, annars false.
  Första fallet används endast under ""map making"*/

/*kollar om man nått målet eller om roboten har gått ett helt varv.
  (X och Y koord är åter (0,0)) och snurrat nästan helt varv*/
//Bör nog ha större felmarginaler (i alla fall i andra fallet)! -Bo
public boolean reachedPos(int x,int y){
    int LIMIT = 3;
    System.out.println("X:"+getX()+"   Y:"+getY());
    if (makeMap)
        return (((getX()>origoX-LIMIT) && (getX()<origoX+LIMIT)) &&
                ((getY()>origoY-LIMIT) && (getY()<origoY+LIMIT)) &&
                ((getAngle()>270) || (getAngle()<-270)));
    else
        return (((getX()>x-LIMIT) && (getX()<x+LIMIT)) &&
                ((getY()>y-LIMIT) && (getY()<y+LIMIT)));
}

public void print() {
    int x = (int)(xCoord/250);
    int y = (int)(yCoord/250);
    System.out.println("Robotens koord: ("+x+","+y+")");
    System.out.println("");
    for (int i=height-1;i>=0;i--){
        System.out.print(i+"  ");
        for (int j=0;j<width;j++) {
            if (map[j][i]==-1)
                System.out.print("#");
            else if (map[j][i]==0)
                System.out.print(" ");
            else if ((j==x)&&(i==y))
                System.out.print("X");
        }
        System.out.println(" "+i);
    }
}
```

```
    public void setMapMaking(boolean b) {
        makeMap = b;
    }

    public int[][] getMap() {
        return map;
    }

    public int[][] getModifiedMap() {
        int i,j,k,l;
        int[][] modifiedMap = new int[width][height];
        for (i=0;i<width;i++)
            for (j=0;j<height;j++)
                modifiedMap[i][j] = map[i][j];

        for (i=1;i<width-1;i++)
            for (j=1;j<height-1;j++)
                if (map[i][j]==-1)
                    for (k=-1;k<=1;k++)
                        for (l=-1;l<=1;l++)
                            if (!((k==0)&&(l==0)))
                                if(map[i+k][j+l]==0)
                                    modifiedMap[i][j]=0;
        return modifiedMap;
    }

}
```